

---

# **cepton\_sdk Documentation**

**Cepton Technologies**

**Oct 11, 2019**



# CONTENTS

<b>1</b>	<b>Overview</b>	<b>1</b>
<b>2</b>	<b>Building</b>	<b>5</b>
<b>3</b>	<b>Samples</b>	<b>7</b>
<b>4</b>	<b>Internal</b>	<b>23</b>
<b>5</b>	<b>API Reference</b>	<b>25</b>
	<b>Index</b>	<b>47</b>



## OVERVIEW

The Cepton SDK provides the following features:

- **Networking:** Listen for sensor packets.
- **Capture Replay:** Read sensor packets from a PCAP file.
- **Parsing:** Parse sensor packets.
- **Calibration:** Apply sensor calibration.
- **Frame Accumulation:** Accumulate sensor points and detect frames.

**Note:** Currently, the Cepton LiDAR packet formats are under active development, and are not publicly available. The SDK is required for **Parsing** and **Calibration**. All other SDK features are optional, and can be done manually by the user.

## 1.1 Getting Started

Below is a very simple SDK usage example. For more complete examples, see [Samples](#).

```
1  #include <cepton_sdk_api.hpp>
2
3  int main(int argc, char **argv) {
4      // Initialize SDK with default options
5      CEPTON_CHECK_ERROR(
6          cepton_sdk::api::initialize(cepton_sdk::create_options(), "", true));
7
8      // Get all sensors
9      for (int i = 0; i < cepton_sdk::get_n_sensors(); ++i) {
10         cepton_sdk::SensorInformation sensor_info;
11         CEPTON_CHECK_ERROR(
12             cepton_sdk::get_sensor_information_by_index(i, sensor_info));
13     }
14
15     // Listen for points
16     cepton_sdk::api::SensorImageFrameCallback callback;
17     CEPTON_CHECK_ERROR(callback.initialize());
18     CEPTON_CHECK_ERROR(
19         callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
20                             const cepton_sdk::SensorImagePoint *c_image_points) {
21             // Get sensor info
22             cepton_sdk::SensorInformation sensor_info;
```

(continues on next page)

(continued from previous page)

```

23     CEPTON_CHECK_ERROR (
24         cepton_sdk::get_sensor_information(handle, sensor_info));
25
26     // Convert points
27     std::vector<cepton_sdk::util::SensorPoint> points(n_points);
28     for (int i = 0; i < n_points; ++i) {
29         cepton_sdk::util::convert_sensor_image_point_to_point (
30             c_image_points[i], points[i]);
31     }
32     });
33
34     // Sleep or do other work...
35
36     // Deinitialize SDK
37     cepton_sdk::deinitialize().ignore();
38 }

```

For prototyping, it is recommended to use the high level C++ API. The C++ API also acts as reference code for C API usage. Any C++ functions that directly wrap C functions are not documented; refer to the C function documentation.

The general C SDK workflow is as follows:

1. Initialize SDK (`cepton_sdk_initialize()`). See *Setup*.
2. Register point listener function (`cepton_sdk_listen_image_frames()`). See *Points*.
3. Wait for sensor calibration/information packets, then query sensors. See *Sensors*.
4. Sleep or run replay. Callbacks will occur asynchronously.
5. Deinitialize SDK.

## 1.2 Packets

The SDK passively listens for sensor UDP packets. There are 2 types of sensor packets:

- **Calibration/Information:** Contains sensor calibration, statistics, and other information. Published at ~1Hz.
- **Points:** Contains ~100 measurements. Published at ~1000Hz.

## 1.3 Errors

Many SDK functions return `CeptonSensorErrorCode`. If this is not `CEPTON_SUCCESS`, then the user must call `cepton_sdk_get_error()`, otherwise the SDK will complain that the error was not checked.

All sensor errors will be returned via `FpCeptonSensorErrorCallback`, which is registered in `cepton_sdk_initialize()`.

## 1.4 Timestamps

All `int64` timestamps are microseconds since the Unix epoch (UTC). All `float` times (measurement period, replay time, frame length, etc.) are time differences measured in seconds. Point timestamps are based on one of the following sources (the first valid source is used):

1. GPS (NMEA + PPS)
2. PTP
3. Host PC

## 1.5 Sensor Fusion

See *Process Single*, *Process Multi*.

## 1.6 Multiple Returns

To enable multiple returns, pass the `CEPTON_SDK_CONTROL_ENABLE_MULTIPLE_RETURNS` flag during initialization.

The returns are as follows:

1. Strongest signal.
2. Furthest signal, if it is not the strongest. Otherwise, the second strongest signal.





## 2.1 CMake

The simplest way to include the SDK is as a CMake subdirectory:

```
add_subdirectory(<cepton_sdk_source_dir>)  
...  
target_link_libraries(<target> cepton_sdk::cepton_sdk)
```

See *CMake*.

## 2.2 Manually

It is also possible to manually link to the SDK as follows:

1. Add the `cepton_sdk_redist/include` path.
2. Link to the correct library binary in `cepton_sdk_redist/lib` or `cepton_sdk_redist/bin`.
3. If using the static library, define `CEPTON_SDK_STATIC`.



## SAMPLES

### 3.1 Building

#### 3.1.1 Unix

```
cd cepton_sdk_redist/samples
mkdir build
cd build
cmake ..
make
```

#### 3.1.2 Windows

The following commands are for a UNIX command line (e.g. Git Bash).

```
cd cepton_sdk_redist/samples
mkdir build
cd build
cmake -G "Visual Studio 15 2017 Win64" ..
```

In Visual Studio:

- Click File -> Open -> Project/Solution. Select `cepton_sdk_redist/samples/build/cepton_sdk_samples.sln`.
- Build the project.

### 3.2 Basic

Start with *Basic* or *C Basic*.

#### 3.2.1 CMake

Listing 1: samples/CMakeLists.txt

```
1 #[[
2 CMake file for building samples.
3 ]]
```

(continues on next page)

(continued from previous page)

```

4 cmake_minimum_required(VERSION 3.1)
5
6 set(CEPTON_SDK_SAMPLES_SOURCE_DIR "${CMAKE_CURRENT_LIST_DIR}")
7 get_filename_component(CEPTON_SDK_SOURCE_DIR
8     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/../" ABSOLUTE)
9 list(APPEND CMAKE_MODULE_PATH "${CEPTON_SDK_SOURCE_DIR}/cmake")
10
11 include("CeptonVersion")
12 project(cepton_sdk_samples VERSION ${CEPTON_VERSION} LANGUAGES C CXX)
13
14 include("CeptonCommon")
15
16 if(GCC OR CLANG)
17     add_flags(-Wall)
18     add_linker_flags(-pthread)
19 endif()
20
21 # cepton_sdk
22 add_subdirectory("${CEPTON_SDK_SOURCE_DIR}"
23     "${PROJECT_BINARY_DIR}/third_party/cepton_sdk")
24
25 set(CEPTON_SDK_SAMPLE_SOURCES
26     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/frame.cpp"
27     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/get_timestamp_offset.cpp"
28     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/process_multi.cpp"
29     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/process_single.cpp"
30     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/advanced/stray.cpp"
31     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/basic.cpp"
32     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/callback.cpp"
33     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/error.cpp"
34     "${CEPTON_SDK_SAMPLES_SOURCE_DIR}/sensor.cpp")
35 foreach(path ${CEPTON_SDK_SAMPLE_SOURCES})
36     get_filename_component(name "${path}" NAME_WE)
37     add_executable(cepton_sdk_sample_${name} "${path}")
38     target_link_libraries(cepton_sdk_sample_${name} cepton_sdk::cepton_sdk)
39 endforeach()

```

### 3.2.2 Basic

Listing 2: samples/basic.cpp

```

1  /**
2   * Sample code for general SDK usage.
3   */
4  #include <vector>
5
6  #include <cepton_sdk_api.hpp>
7
8  #include "common.hpp"
9
10 /// Sample points callback.
11 class FramesListener {
12 public:
13     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
14                         const cepton_sdk::SensorImagePoint *c_image_points) {

```

(continues on next page)

(continued from previous page)

```

15 // Get sensor info
16 cepton_sdk::SensorInformation sensor_info;
17 CEPTON_CHECK_ERROR(cepton_sdk::get_sensor_information(handle, sensor_info));
18
19 // Convert points
20 static thread_local std::vector<cepton_sdk::util::SensorPoint> points;
21 points.resize(n_points);
22 for (int i = 0; i < n_points; ++i) {
23     cepton_sdk::util::convert_sensor_image_point_to_point(c_image_points[i],
24                                                         points[i]);
25 }
26
27 // Print
28 if (i_frame < 5) {
29     std::printf("Received %i points from sensor %i\n", (int)n_points,
30               (int)sensor_info.serial_number);
31 }
32 ++i_frame;
33 }
34
35 private:
36     std::size_t i_frame = 0;
37 };
38
39 int main(int argc, char **argv) {
40     // Parse arguments
41     check_help(argc, argv, "cepton_sdk_sample_basic [capture_path]");
42     std::string capture_path;
43     if (argc >= 2) capture_path = argv[1];
44
45     std::printf("Press Ctrl+C to stop\n");
46
47     // Initialize SDK
48     auto options = cepton_sdk::create_options();
49
50     // By default, return points every packet.
51
52     // Uncomment to return points every frame.
53     // options.frame.mode = CEPTON_SDK_FRAME_COVER;
54
55     // Uncomment to return points at fixed time interval.
56     // options.frame.mode = CEPTON_SDK_FRAME_TIMED;
57     // options.frame.length = 0.1f;
58
59     // Wait short duration for sensors to connect.
60     const bool enable_wait = true;
61
62     std::printf("Initializing...\n");
63     CEPTON_CHECK_ERROR(
64         cepton_sdk::api::initialize(options, capture_path, enable_wait));
65
66     // Get all sensors
67     const int n_sensors = (int)cepton_sdk::get_n_sensors();
68     for (int i = 0; i < n_sensors; ++i) {
69         cepton_sdk::SensorInformation sensor_info;
70         CEPTON_CHECK_ERROR(
71             cepton_sdk::get_sensor_information_by_index(i, sensor_info));

```

(continues on next page)

(continued from previous page)

```

72     std::printf("Sensor: %i\n", (int)sensor_info.serial_number);
73 }
74
75 // Listen for points
76 std::printf("Listening for points...\n");
77 cepton_sdk::api::SensorImageFrameCallback callback;
78 CEPTON_CHECK_ERROR(callback.initialize());
79 FramesListener frames_listener;
80 CEPTON_CHECK_ERROR(
81     callback.listen(&frames_listener, &FramesListener::on_image_frame));
82
83 // Run (sleep or run replay)
84 CEPTON_CHECK_ERROR(cepton_sdk::api::wait());
85
86 // Deinitialize
87 cepton_sdk::deinitialize().ignore();
88 }

```

### 3.2.3 C Basic

Listing 3: samples/c\_basic.c

```

1  /**
2   * Sample code for general C SDK usage.
3   */
4  #include <stdio.h>
5  #include <time.h>
6
7  #include <cepton_sdk.h>
8
9  void check_sdk_error() {
10     const char *error_msg;
11     const auto error_code = cepton_sdk_get_error(&error_msg);
12     printf("%s: %s\n", cepton_get_error_code_name(error_code), error_msg);
13     exit(1);
14 }
15
16 int frames_got = 0;
17 time_t first_frame_time = 0;
18 time_t current_frame_time = 0;
19
20 void image_frame_callback(CeptonSensorHandle handle, size_t n_points,
21     const struct CeptonSensorImagePoint *c_points,
22     void *user_data) {
23     // Get sensor info
24     struct CeptonSensorInformation sensor_info;
25     cepton_sdk_get_sensor_information(handle, &sensor_info);
26     check_sdk_error();
27 }
28
29 int main() {
30     // Initialize
31     struct CeptonSDKOptions options = cepton_sdk_create_options();
32     cepton_sdk_initialize(CEPTON_SDK_VERSION, &options, NULL, NULL);
33     check_sdk_error();

```

(continues on next page)

(continued from previous page)

```

34
35 // Wait for sensor
36 const int n_sensors = (int) cepton_sdk_get_n_sensors();
37 while (!cepton_sdk_get_n_sensors())
38     ;
39 for (int i = 0; i < n_sensors; ++i) {
40     struct CeptonSensorInformation sensor_info;
41     cepton_sdk_get_sensor_information_by_index(0, &sensor_info);
42     check_sdk_error();
43     printf("Sensor: %i\n", (int) sensor_info.serial_number);
44 }
45
46 // Listen for frames
47 cepton_sdk_listen_image_frames(image_frame_callback, NULL);
48 check_sdk_error();
49
50 // Sleep
51 while (frames_got < 100)
52     ;
53
54 // Deinitialize
55 cepton_sdk_deinitialize();
56 check_sdk_error();
57 }

```

## 3.2.4 Callback

Listing 4: samples/callback.cpp

```

1  /**
2   * Sample code for callback usage.
3   */
4  #include <cepton_sdk_api.hpp>
5
6  // Sample global callback.
7  void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
8                     const cepton_sdk::SensorImagePoint *c_image_points) {}
9
10 // Sample member callback.
11 class FramesListener {
12 public:
13     void on_image_frame(cepton_sdk::SensorHandle handle, std::size_t n_points,
14                        const cepton_sdk::SensorImagePoint *c_image_points) {}
15 };
16
17 int main(int argc, char **argv) {
18     // Initialize
19     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize());
20     cepton_sdk::api::SensorImageFrameCallback callback;
21     CEPTON_CHECK_ERROR(callback.initialize());
22
23     // Listen lambda
24     CEPTON_CHECK_ERROR(callback.listen(
25         [] (cepton_sdk::SensorHandle handle, std::size_t n_points,
26            const cepton_sdk::SensorImagePoint *c_image_points) {}));
26 }

```

(continues on next page)

(continued from previous page)

```

27
28 // Listen global function
29 CEPTON_CHECK_ERROR(callback.listen(on_image_frame));
30
31 // Listen member function
32 FramesListener frames_listener;
33 CEPTON_CHECK_ERROR(
34     callback.listen(&frames_listener, &FramesListener::on_image_frame));
35
36 // Deinitialize
37 cepton_sdk::deinitialize().ignore();
38 }

```

## 3.2.5 Error

Listing 5: samples/error.cpp

```

1 /**
2  * Sample code for error callback usage.
3  */
4 #include <cepton_sdk_api.hpp>
5
6 int main(int argc, char** argv) {
7     // Initialize
8     cepton_sdk::api::SensorErrorCallback error_callback;
9     CEPTON_CHECK_ERROR(error_callback.listen(
10         [&](cepton_sdk::SensorHandle handle,
11             const cepton_sdk::SensorError& error) { throw error; }));
12     CEPTON_CHECK_ERROR(cepton_sdk::initialize(
13         CEPTON_SDK_VERSION, cepton_sdk::create_options(),
14         error_callback.global_on_callback, &error_callback));
15
16     // Deinitialize
17     cepton_sdk::deinitialize().ignore();
18 }

```

## 3.3 Advanced

### 3.3.1 Frame

Listing 6: samples/advanced/frame.cpp

```

1 /**
2  * Sample code for custom frame accumulation.
3  */
4
5 #include <cepton_sdk_api.hpp>
6
7 int main(int argc, char **argv) {
8     std::string capture_path;
9     if (argc >= 2) capture_path = argv[1];

```

(continues on next page)



(continued from previous page)

```

10
11  auto frame_options = cepton_sdk::create_frame_options();
12
13  // Cover
14  frame_options.mode = CEPTON_SDK_FRAME_COVER;
15
16  // Cycle
17  // frame_options.mode = CEPTON_SDK_FRAME_CYCLE;
18
19  // Timed
20  // frame_options.mode = CEPTON_SDK_FRAME_TIMED;
21  // frame_options.length = 0.1f;
22
23  // Initialize
24  auto options = cepton_sdk::create_options();
25  CEPTON_CHECK_ERROR(ception_sdk::api::initialize(options, capture_path));
26  cepton_sdk::api::SensorImageFrameCallback callback;
27  CEPTON_CHECK_ERROR(callback.initialize());
28
29  // Get sensor
30  while (ception_sdk::get_n_sensors() == 0)
31      CEPTON_CHECK_ERROR(ception_sdk::api::wait(0.1f));
32  cepton_sdk::SensorInformation sensor_info;
33  CEPTON_CHECK_ERROR(
34      cepton_sdk::get_sensor_information_by_index(0, sensor_info));
35
36  // Create accumulator
37  cepton_sdk::util::FrameAccumulator accumulator(sensor_info);
38  CEPTON_CHECK_ERROR(accumulator.set_options(frame_options));
39  CEPTON_CHECK_ERROR(callback.listen(
40      [&](ception_sdk::SensorHandle handle, std::size_t n_points,
41          const cepton_sdk::SensorImagePoint *const c_image_points) {
42          if (handle != sensor_info.handle) return;
43          accumulator.add_points(n_points, c_image_points);
44      }));
45
46  // Listen
47  CEPTON_CHECK_ERROR(accumulator.callback.listen(
48      [&](std::size_t n_points,
49          const cepton_sdk::SensorImagePoint *const c_image_points) {
50          const float frame_length =
51              cepton_sdk::util::from_usec(c_image_points[n_points - 1].timestamp -
52              c_image_points[0].timestamp);
53          std::printf("Received %i points (%.3fs)\n", (int)n_points,
54              frame_length);
55      }));
56
57  // Run
58  CEPTON_CHECK_ERROR(ception_sdk::api::wait(5.0f));
59
60  // Deinitialize
61  cepton_sdk::deinitialize().ignore();
62  }

```

### 3.3.2 Network

Listing 7: samples/advanced/network.cpp

```

1  /**
2   * Sample code for custom networking.
3   */
4  #include <asio.hpp>
5
6  #include <cepton_sdk_api.hpp>
7
8  using asio::ip::udp;
9
10 class SocketListener {
11 public:
12     SocketListener() : m_socket(m_io_service, udp::v4()) {
13         m_socket.set_option(asio::socket_base::reuse_address(true));
14         m_socket.bind(udp::endpoint(udp::v4(), 8808));
15     }
16
17     void run() {
18         listen();
19         m_io_service.run_for(std::chrono::seconds(5));
20     }
21
22     void listen() {
23         m_socket.async_receive_from(
24             asio::buffer(m_buffer), m_end_point,
25             [this](const asio::error_code& error, std::size_t buffer_size) {
26                 if (buffer_size == 0) return;
27                 if (error == asio::error::operation_aborted) return;
28                 const CeptonSensorHandle handle =
29                     m_end_point.address().to_v4().to_ulong();
30                 // For more accurate timestamps, a separate network receive thread
31                 // should be
32                 // used.
33                 const int64_t timestamp = cepton_sdk::util::get_timestamp_usec();
34                 CEPTON_CHECK_ERROR(cepton_sdk::mock_network_receive(
35                     handle, timestamp, m_buffer.data(), buffer_size));
36                 listen();
37             });
38     }
39
40 private:
41     asio::io_context m_io_service;
42     udp::socket m_socket;
43     udp::endpoint m_end_point;
44     std::array<uint8_t, 4096> m_buffer;
45 };
46
47 int main() {
48     // Initialize
49     auto options = cepton_sdk::create_options();
50     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
51     options.frame.mode = CEPTON_SDK_FRAME_COVER;
52     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));
53
54     // Listen for points
55     cepton_sdk::api::SensorImageFrameCallback callback;

```

(continues on next page)

(continued from previous page)

```

56  CEPTON_CHECK_ERROR(callback.initialize());
57  CEPTON_CHECK_ERROR(
58      callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
59                          const cepton_sdk::SensorImagePoint* c_image_points) {
60          std::printf("Received %i points from sensor %lli\n", (int)n_points,
61                      (long long)handle);
62      }));
63
64  SocketListener listener;
65  listener.run();
66
67  // Deinitialize
68  cepton_sdk::deinitialize().ignore();
69  }

```

### 3.3.3 Organize Points

Listing 8: samples/advanced/organize\_points.cpp

```

1  /*****
2  **
3  ** Copyright(C) 2019 Cepton Technologies. All Rights Reserved. **
4  ** Contact: https://www.cepton.com **
5  ** **
6  ** Sample code which opens a cepton sensor pcap file, organizes **
7  ** the points and continuously saves the most recent organized **
8  ** points to a frame to a cvs file "organized_cloud.cvs **
9  *****/
10
11 #include <cepton_sdk_util.hpp>
12 #include <cepton_sdk/capture.hpp>
13 #include <cepton_sdk_api.hpp>
14
15 using namespace cepton_sdk::util;
16
17 int main(int argc, char** argv) {
18     if (argc < 2) return -1;
19     const std::string capture_path = argv[1];
20
21     // Initialize sdk
22     auto options = cepton_sdk::create_options();
23     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
24     options.frame.mode = CEPTON_SDK_FRAME_COVER;
25     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));
26
27     cepton_sdk::SensorInformation sensor_info;
28
29     OrganizedCloud organized_cloud;
30     std::ofstream os;
31
32     cepton_sdk::Capture m_capture;
33     CEPTON_CHECK_ERROR(m_capture.open_for_read(capture_path));
34
35     CEPTON_CHECK_ERROR(
36         cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,

```

(continues on next page)

(continued from previous page)

```

37         CEPTON_SDK_CONTROL_DISABLE_NETWORK));
38 CEPTON_CHECK_ERROR(cepton_sdk_clear());
39
40 // Listen for points
41 cepton_sdk::api::SensorImageFrameCallback callback;
42 CEPTON_CHECK_ERROR(callback.initialize());
43 callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
44                  const cepton_sdk::SensorImagePoint* c_image_points) {
45
46     cepton_sdk::get_sensor_information(handle, sensor_info);
47
48     std::printf("Received %i points from sensor %lli\n", static_cast<int>(n_points),
49               static_cast<long long>(handle));
50
51     Organizer organizer(sensor_info);
52
53     organizer.organize_points(n_points,
54                             sensor_info.return_count,
55                             c_image_points,
56                             organized_cloud);
57
58     os.open("organize_cloud.csv");
59     for (const auto& point : organized_cloud.points)
60     {
61         if (point.valid)
62         {
63             float x = 0;
64             float y = 0;
65             float z = 0;
66             cepton_sdk::util::convert_image_point_to_point(
67                 point.image_x, point.image_z, point.distance, x,
68                 y, z);
69
70             os << x << "," << y << "," << z << "\n";
71         }
72     }
73     os.close();
74 });
75
76 while (true) {
77     cepton_sdk::Capture::PacketHeader header;
78     const uint8_t* data;
79     CEPTON_CHECK_ERROR(m_capture.next_packet(header, data));
80
81     const cepton_sdk::SensorHandle handle =
82         static_cast<cepton_sdk::SensorHandle>(header.ip_v4) |
83         CEPTON_SENSOR_HANDLE_FLAG MOCK;
84     CEPTON_CHECK_ERROR(cepton_sdk_mock_network_receive(
85         handle, header.timestamp, data, static_cast<size_t>(header.data_size)));
86 }
87 }

```

### 3.3.4 Process Multi

Listing 9: samples/advanced/process\_multi.cpp

```

1  /**
2   * Sample code for processing data from multiple sensors.
3   */
4  #include <string>
5  #include <vector>
6
7  #include <cepton_sdk_api.hpp>
8
9  struct Frame {
10     int64_t timestamp;
11     std::map<cepton_sdk::SensorHandle, std::vector<cepton_sdk::SensorImagePoint>>
12         image_points_dict;
13 };
14
15 class FrameAccumulator {
16 public:
17     void on_image_frame(
18         cepton_sdk::SensorHandle handle, std::size_t n_points,
19         const cepton_sdk::SensorImagePoint* const c_image_points) {
20         cepton_sdk::util::LockGuard lock(m_mutex);
21
22         auto& image_points = m_image_points_dict[handle];
23         image_points.reserve(image_points.size() + n_points);
24         image_points.insert(image_points.end(), c_image_points,
25                             c_image_points + n_points);
26
27         check_and_publish();
28     }
29
30 private:
31     void check_and_publish() {
32         const auto timestamp = cepton_sdk::api::get_time();
33
34         if (timestamp < m_timestamp) return;
35         if ((timestamp - m_timestamp) < int64_t(m_frame_length * 1e6f)) return;
36         m_timestamp = timestamp;
37
38         auto frame = std::make_shared<Frame>();
39         frame->timestamp = timestamp;
40         frame->image_points_dict = m_image_points_dict;
41         m_image_points_dict.clear();
42         queue.push(frame);
43     }
44
45 public:
46     cepton_sdk::util::SingleConsumerQueue<Frame> queue;
47
48 private:
49     std::timed_mutex m_mutex;
50     float m_frame_length = 0.1f;
51     int64_t m_timestamp = 0;
52     std::map<cepton_sdk::SensorHandle, std::vector<cepton_sdk::SensorImagePoint>>
53         m_image_points_dict;
54 };
55

```

(continues on next page)

(continued from previous page)

```

56 int main(int argc, char** argv) {
57     std::string capture_path;
58     if (argc >= 2) capture_path = argv[1];
59
60     // Initialize
61     auto options = cepton_sdk::create_options();
62     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
63     cepton_sdk::api::SensorImageFrameCallback callback;
64     CEPTON_CHECK_ERROR(callback.initialize());
65     if (cepton_sdk::capture_replay::is_open())
66         CEPTON_CHECK_ERROR(cepton_sdk::capture_replay::resume());
67
68     // Listen
69     FrameAccumulator accumulator;
70     CEPTON_CHECK_ERROR(
71         callback.listen(&accumulator, &FrameAccumulator::on_image_frame));
72     while (true) {
73         const auto frame = accumulator.queue.pop(0.01f);
74         if (!frame) continue;
75         // Do processing
76     }
77
78     // Deinitialize
79     cepton_sdk::deinitialize().ignore();
80 }

```

### 3.3.5 Process Single

Listing 10: samples/advanced/process\_single.cpp

```

1  /**
2   * Sample code for processing data from a single sensor.
3   */
4  #include <memory>
5  #include <vector>
6
7  #include <cepton_sdk_api.hpp>
8
9  struct Frame {
10     int64_t timestamp;
11     cepton_sdk::SensorHandle handle;
12     std::vector<cepton_sdk::SensorImagePoint> image_points;
13 };
14
15 int main(int argc, char **argv) {
16     std::string capture_path;
17     if (argc >= 2) capture_path = argv[1];
18
19     // Initialize
20     auto options = cepton_sdk::create_options();
21     options.frame.mode = CEPTON_SDK_FRAME_TIMED;
22     options.frame.length = 0.1f;
23     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options, capture_path));
24     cepton_sdk::api::SensorImageFrameCallback callback;
25     CEPTON_CHECK_ERROR(callback.initialize());

```

(continues on next page)

(continued from previous page)

```

26  if (cepton_sdk::capture_replay::is_open())
27      CEPTON_CHECK_ERROR(cepton_sdk::capture_replay::resume());
28
29  // Listen
30  cepton_sdk::util::SingleConsumerQueue<Frame> queue;
31  CEPTON_CHECK_ERROR(
32      callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
33                          const cepton_sdk::SensorImagePoint *c_image_points) {
34          auto frame = std::make_shared<Frame>();
35          frame->timestamp = cepton_sdk::api::get_time();
36          frame->handle = handle;
37          frame->image_points.reserve(n_points);
38          frame->image_points.insert(frame->image_points.end(), c_image_points,
39                                   c_image_points + n_points);
40          queue.push(frame);
41      }));
42  while (true) {
43      const auto frame = queue.pop(0.01f);
44      if (!frame) continue;
45      // Do processing
46  }
47
48  // Deinitialize
49  cepton_sdk::deinitialize().ignore();
50 }

```

### 3.3.6 Replay

Listing 11: samples/advanced/replay.cpp

```

1  /**
2   * Sample code for custom packet replaying.
3   */
4  #include <cepton_sdk/capture.hpp>
5  #include <cepton_sdk_api.hpp>
6
7  class CaptureReplay {
8  public:
9      CaptureReplay(const std::string& path) {
10          CEPTON_CHECK_ERROR(m_capture.open_for_read(path));
11
12          CEPTON_CHECK_ERROR(
13              cepton_sdk_set_control_flags(CEPTON_SDK_CONTROL_DISABLE_NETWORK,
14                                           CEPTON_SDK_CONTROL_DISABLE_NETWORK));
15          CEPTON_CHECK_ERROR(cepton_sdk_clear());
16      }
17
18      ~CaptureReplay() {
19          m_capture.close();
20          if (cepton_sdk_is_initialized()) {
21              CEPTON_CHECK_ERROR(cepton_sdk_clear());
22          }
23      }
24
25      void run() {

```

(continues on next page)

(continued from previous page)

```

26     while (true) {
27         cepton_sdk::Capture::PacketHeader header;
28         const uint8_t* data;
29         CEPTON_CHECK_ERROR(m_capture.next_packet(header, data));
30
31         const cepton_sdk::SensorHandle handle =
32             (cepton_sdk::SensorHandle)header.ip_v4 |
33             CEPTON_SENSOR_HANDLE_FLAG MOCK;
34         CEPTON_CHECK_ERROR(cepton_sdk_mock_network_receive(
35             handle, header.timestamp, data, header.data_size));
36     }
37 }
38
39 private:
40     cepton_sdk::Capture m_capture;
41 };
42
43 int main(int argc, char** argv) {
44     if (argc < 2) return -1;
45     const std::string capture_path = argv[1];
46
47     // Initialize
48     auto options = cepton_sdk::create_options();
49     options.control_flags |= CEPTON_SDK_CONTROL_DISABLE_NETWORK;
50     options.frame.mode = CEPTON_SDK_FRAME_COVER;
51     CEPTON_CHECK_ERROR(cepton_sdk::api::initialize(options));
52
53     // Listen for points
54     cepton_sdk::api::SensorImageFrameCallback callback;
55     CEPTON_CHECK_ERROR(callback.initialize());
56     CEPTON_CHECK_ERROR(
57         callback.listen([](cepton_sdk::SensorHandle handle, std::size_t n_points,
58             const cepton_sdk::SensorImagePoint* c_image_points) {
59             std::printf("Received %i points from sensor %lli\n", (int)n_points,
60                 (long long)handle);
61         }));
62
63     // Run
64     CaptureReplay replay(capture_path);
65     replay.run();
66
67     // Deinitialize
68     cepton_sdk::deinitialize().ignore();
69 }

```

### 3.3.7 Stray

Listing 12: samples/advanced/stray.cpp

```

1  /**
2   * Sample code for stray filter usage.
3   */
4  #include <vector>
5
6  #include <cepton_sdk_api.hpp>

```

(continues on next page)



(continued from previous page)

```
7
8 int main(int argc, char **argv) {
9     // Initialize
10    CEPTON_CHECK_ERROR(cepton_sdk::api::initialize());
11
12    cepton_sdk::api::SensorImageFrameCallback callback;
13    CEPTON_CHECK_ERROR(callback.initialize());
14    cepton_sdk::util::StrayFilter filter;
15    CEPTON_CHECK_ERROR(
16        callback.listen([&](cepton_sdk::SensorHandle handle, std::size_t n_points,
17                            const cepton_sdk::SensorImagePoint *c_image_points) {
18            // Get sensor
19            cepton_sdk::SensorInformation sensor_info;
20            CEPTON_CHECK_ERROR(
21                cepton_sdk::get_sensor_information(handle, sensor_info));
22
23            // Copy points to buffer
24            std::vector<cepton_sdk::SensorImagePoint> image_points;
25            image_points.insert(image_points.begin(), c_image_points,
26                               c_image_points + n_points);
27
28            // Filter stray
29            filter.init(sensor_info);
30            filter.run((int)n_points, image_points.data());
31        }));
32
33    // Run
34    CEPTON_CHECK_ERROR(cepton_sdk::api::wait(5.0f));
35
36    // Deinitialize
37    cepton_sdk::deinitialize().ignore();
38 }
```



## INTERNAL

This page provides a brief description of what goes on inside the SDK.

### 4.1 Packet Received

Occurs when `cepton_sdk_mock_network_receive` is called.

1. **If the packet is a sensor information packet:**

- Update the internal stored sensor state. This information can be queried with `cepton_sdk_get_sensor_information()`.

2. **If the packet is a data packet:**

- If no corresponding sensor information packet has been received, ignore the data packet.
- Compute points from packet.
- Apply sensor calibration.
- Add points to frame accumulator. If frame is complete, emit image frame callback.

### 4.2 Threads

#### 4.2.1 Networking Thread 0

Created if `CEPTON_SDK_CONTROL_DISABLE_NETWORK` is not set.

- Start listening on the UDP port (default: 8808).
- Push received packets on a queue for Thread 1.

#### 4.2.2 Networking Thread 1

Created if `CEPTON_SDK_CONTROL_DISABLE_NETWORK` is not set.

- Pop packets from queue.
- Call internal version of `cepton_sdk_mock_network_receive`. See `Packet Received`.

### 4.2.3 Capture Replay Thread 0

Created by `cepton_sdk_capture_replay_resume()`.

- **While running**
  - Read next packet from PCAP file.
  - Sleep to simulate realtime delays.
  - Call `cepton_sdk_mock_network_receive`.

## 4.3 Concurrency

All SDK getter functions are thread safe, and can be called from callbacks. Other SDK functions are not guaranteed to be thread safe, and can cause deadlock if called from callbacks.

## 4.4 Minimal SDK

If desired, the following SDK features can be disabled in the SDK and performed manually by the user:

- Capture Replay: *Replay*.
- Frame Accumulation: *Frame*.
- Networking: *Network*.

## API REFERENCE

### 5.1 Errors

#### 5.1.1 Types

**class SensorError** : public runtime\_error

Error returned by most functions.

Implicitly convertible from/to SensorErrorCode. Getter functions do not return an error, because they cannot fail. Will call CEPTON\_ASSERT if nonzero error is not used (call ignore to manually use error).

#### Public Functions

**SensorError** (SensorErrorCode *code*, const std::string &*msg*)

Create *SensorError* class object with SensorErrorCode and error message.

**SensorError** (SensorErrorCode *code*)

Create *SensorError* class object with error code.

**SensorError** ()

*SensorError* class default constructor.

**~SensorError** ()

*SensorError* class destructor.

**SensorError** (const *SensorError* &*other*)

Create *SensorError* object using *SensorError* object.

*SensorError* &**operator=** (const *SensorError* &*other*)

*SensorError* class assignment operator.

bool **used** () const

Internal use only.

const *SensorError* &**ignore** () const

Mark error as used.

const char \***what** () const

const std::string &**msg** () const

Returns error message.

**SensorErrorCode code () const**

Returns error code.

**operator SensorErrorCode () const**

Implicitly convert to SensorErrorCode.

**operator bool () const**

Returns `false` if error code is `CEPTON_SUCCESS`, `true` otherwise.

**const std::string name () const**

Returns error code name.

**bool is\_error () const**

Returns true if parent object is an error.

**bool is\_fault () const**

Returns true if parent object is a fault.

**typedef int32\_t CeptonSensorErrorCode**

Error code returned by most library functions.

Must call `cepton_sdk_get_error` if nonzero error code is returned.

**enum \_CeptonSensorErrorCode**

*Values:*

**CEPTON\_SUCCESS = 0**

No error.

**CEPTON\_ERROR\_GENERIC = -1**

Generic error.

**CEPTON\_ERROR\_OUT\_OF\_MEMORY = -2**

Failed to allocate heap memory.

**CEPTON\_ERROR\_SENSOR\_NOT\_FOUND = -4**

Could not find sensor.

**CEPTON\_ERROR\_SDK\_VERSION\_MISMATCH = -5**

SDK version mismatch.

**CEPTON\_ERROR\_COMMUNICATION = -6**

Networking error.

**CEPTON\_ERROR\_TOO\_MANY\_CALLBACKS = -7**

Callback already set.

**CEPTON\_ERROR\_INVALID\_ARGUMENTS = -8**

Invalid value or uninitialized struct.

**CEPTON\_ERROR\_ALREADY\_INITIALIZED = -9**

Already initialized.

**CEPTON\_ERROR\_NOT\_INITIALIZED = -10**

Not initialized.

**CEPTON\_ERROR\_INVALID\_FILE\_TYPE = -11**

Invalid file type.

**CEPTON\_ERROR\_FILE\_IO = -12**

File IO error.

**CEPTON\_ERROR\_CORRUPT\_FILE** = -13  
Corrupt/invalid file.

**CEPTON\_ERROR\_NOT\_OPEN** = -14  
Not open.

**CEPTON\_ERROR\_EOF** = -15  
End of file.

**CEPTON\_FAULT\_INTERNAL** = -1000  
Internal sensor parameter out of range.

**CEPTON\_FAULT\_EXTREME\_TEMPERATURE** = -1001  
Extreme sensor temperature fault.

**CEPTON\_FAULT\_EXTREME\_HUMIDITY** = -1002  
Extreme sensor humidity fault.

**CEPTON\_FAULT\_EXTREME\_ACCELERATION** = -1003  
Extreme sensor acceleration fault.

**CEPTON\_FAULT\_ABNORMAL\_FOV** = -1004  
Abnormal sensor FOV fault.

**CEPTON\_FAULT\_ABNORMAL\_FRAME\_RATE** = -1005  
Abnormal sensor frame rate fault.

**CEPTON\_FAULT\_MOTOR\_MALFUNCTION** = -1006  
Sensor motor malfunction fault.

**CEPTON\_FAULT\_LASER\_MALFUNCTION** = -1007  
Sensor laser malfunction fault.

**CEPTON\_FAULT\_DETECTOR\_MALFUNCTION** = -1008  
Sensor detector malfunction fault.

### 5.1.2 Methods

**const char \***`cepton_get_error_code_name` (*CeptonSensorErrorCode error\_code*)  
Returns error code name string.

Returns empty string if error code is invalid.

**Return** Error code name string. Owned by SDK. Valid until next SDK call in current thread.

**int** `cepton_is_error_code` (*CeptonSensorErrorCode error\_code*)  
Returns whether `error_code` is of the form `CEPTON_ERROR_*`.

**int** `cepton_is_fault_code` (*CeptonSensorErrorCode error\_code*)  
Returns whether `error_code` is of the form `CEPTON_FAULT_*`.

*CeptonSensorErrorCode* `cepton_sdk_get_error` (**const** char \*\**error\_msg*)  
Returns and clears last sdk error.

**Return** Error code.

**Parameters**

- `error_msg`: Returned error message string. Owned by the SDK, and valid until next SDK call in the current thread.

## 5.2 Setup

### 5.2.1 Types

**typedef** uint32\_t **CeptonSDKControl**

SDK setup flags.

**enum** **\_CeptonSDKControl**

*Values:*

**CEPTON\_SDK\_CONTROL\_DISABLE\_NETWORK** = 1 << 1

Disable networking operations.

Useful for running multiple instances of sdk in different processes. Must pass packets manually to `cepton_sdk::mock_network_receive`.

**CEPTON\_SDK\_CONTROL\_DISABLE\_IMAGE\_CLIP** = 1 << 2

Disable marking image clipped points as invalid.

Internal use. Does not affect number of points returned.

**CEPTON\_SDK\_CONTROL\_DISABLE\_DISTANCE\_CLIP** = 1 << 3

Disable marking distance clipped points as invalid.

Internal use. Does not affect number of points returned.

**CEPTON\_SDK\_CONTROL\_ENABLE\_MULTIPLE\_RETURNS** = 1 << 4

Enable multiple returns.

When set, `cepton_sdk::SensorInformation::return_count` will indicate the number of returns per laser. Can only be set at SDK initialization.

**CEPTON\_SDK\_CONTROL\_ENABLE\_STRAY\_FILTER** = 1 << 5

Enable marking stray points as invalid (measurement noise).

Uses `cepton_sdk::util::StrayFilter` to mark points invalid.

Does not affect number of points returned.

**CEPTON\_SDK\_CONTROL\_HOST\_TIMESTAMPS** = 1 << 6

Always use packet timestamps (disable GPS/PTP timestamps).

**CEPTON\_SDK\_CONTROL\_ENABLE\_CROSSTALK\_FILTER** = 1 << 7

Enable marking crosstalk points as invalid.

Not implemented.

**typedef** uint32\_t **CeptonSDKFrameMode**

Controls frequency of points being reported.

**enum** **\_CeptonSDKFrameMode**

*Values:*

**CEPTON\_SDK\_FRAME\_STREAMING** = 0

Report points by packet.

**CEPTON\_SDK\_FRAME\_TIMED** = 1

Report points at fixed time intervals.

Interval controlled by `CeptonSDKFrameOptions::length`.

**CEPTON\_SDK\_FRAME\_COVER** = 2

Report points when the field of view is covered once.



Use this for a fast frame rate.

- For Sora series, detects scanline (left-to-right or right-to-left).
- For HR80 series, detects half scan cycle (left-to-right or right-to-left).
- For Vista series, detects half scan cycle.

**CEPTON\_SDK\_FRAME\_CYCLE = 3**

Report points when the scan pattern goes through a full cycle.

Use this for a consistent, repeating frame. Typically 2x longer frame than `CEPTON_SDK_FRAME_COVER` mode.

**CEPTON\_SDK\_FRAME\_MODE\_MAX = 3**

**struct CeptonSDKFrameOptions**

SDK frame options.

Must use `cepton_sdk_create_frame_options` to create.

### Public Members

size\_t **signature**

Internal use only.

*CeptonSDKFrameMode* **mode**

Default: `CEPTON_SDK_FRAME_STREAMING`.

float **length**

Frame length [seconds].

Default: 0.05. Only used if `mode=CEPTON_SDK_FRAME_TIMED`.

**struct *CeptonSDKFrameOptions* cepton\_sdk\_create\_frame\_options()**

Create frame options.

**struct CeptonSDKOptions**

SDK initialization options.

Must call `cepton_sdk_create_options` to create.

### Public Members

size\_t **signature**

Internal use only.

*CeptonSDKControl* **control\_flags**

Default: 0.

**struct *CeptonSDKFrameOptions* frame**

uint16\_t **port**

Network listen port. Default: 8808.

**struct *CeptonSDKOptions* cepton\_sdk\_create\_options()**

Create SDK options.

**typedef void (\*cepton\_sdk::FpSensorErrorCallback)** (SensorHandle handle, SensorErrorCode error\_code, **const** char \*error\_msg, **const** void \*error\_data, size\_t error\_data\_size, void \*user\_data)

## 5.2.2 Methods

**const char \***`cepton_sdk_get_version_string()`

Returns library version string.

This is different from `CEPTON_SDK_VERSION`.

**Return** Version string. Owned by SDK. Valid until next SDK call in current thread.

**int** `cepton_sdk_get_version_major()`

Returns library version major.

**int** `cepton_sdk_get_version_minor()`

Returns library version minor.

*CeptonSensorErrorCode* `cepton_sdk_initialize`(**int** *ver*, **const struct** *CeptonSDKOptions* \***const** *options*, **FpCeptonSensorErrorCallback** *cb*, **void \*****const** *user\_data*)

Initializes settings and networking.

Must be called before any other sdk function listed below.

### Parameters

- *ver*: `CEPTON_SDK_VERSION`
- *options*: SDK options.
- *cb*: Error callback.
- *user\_data*: Error callback user instance pointer.

*CeptonSensorErrorCode* `cepton_sdk_deinitialize()`

Resets everything and deallocates memory.

*CeptonSensorErrorCode* `cepton_sdk_clear()`

Clears sensors.

Use when loading/unloading capture file.

*CeptonSensorErrorCode* `cepton_sdk_set_control_flags`(*CeptonSDKControl* *mask*, *CeptonSDKControl* *flags*)

Sets SDK control flags.

### Parameters

- *mask*: Bit mask for selecting flags to change.
- *flags*: Bit flag values.

*CeptonSDKControl* `cepton_sdk_get_control_flags()`

Returns SDK control flag.

**int** `cepton_sdk_has_control_flag`(*CeptonSDKControl* *flag*)

Returns whether SDK control flag is set.

**uint16\_t** `cepton_sdk_get_port()`

Returns network listen port.

*CeptonSensorErrorCode* `cepton_sdk_set_port`(**uint16\_t** *port*)

Sets network listen port.

Default: 8808.

*CeptonSensorErrorCode* **cepton\_sdk\_set\_frame\_options**(const struct *CeptonSDKFrameOptions* \*const options)

Sets frame options.

*CeptonSDKFrameMode* **cepton\_sdk\_get\_frame\_mode**()

Returns frame mode.

float **cepton\_sdk\_get\_frame\_length**()

Returns frame length.

## 5.3 Sensors

### 5.3.1 Types

**typedef uint64\_t CeptonSensorHandle**

Sensor identifier.

Generated from sensor IP address.

**typedef uint16\_t CeptonSensorModel**

Sensor model.

**enum \_CeptonSensorModel**

Values:

**HR80W** = 3

**HR80T\_R2** = 6

**VISTA\_860\_GEN2** = 7

**VISTA\_X120** = 10

**SORA\_P60** = 11

**VISTA\_P60** = 12

**VISTA\_X15** = 13

**VISTA\_P90** = 14

**SORA\_P90** = 15

**VISTA\_P61** = 16

**SORA\_P61** = 17

**VISTA\_H120** = 18

**CEPTON\_SENSOR\_MODEL\_MAX** = 18

**struct CeptonSensorInformation**

Sensor information struct.

Returned by `cepton_sdk_get_sensor_information*`.

### Public Members

*CeptonSensorHandle* **handle**

Sensor identifier (generated from IP address).

`uint64_t serial_number`  
Sensor serial number.

`char model_name[28]`  
Full sensor model name.

*CeptonSensorModel* `model`  
Sensor model.

`uint16_t reserved`

`char firmware_version[28]`  
Firmware version string.

`uint8_t major`  
Major firmware version.

`uint8_t minor`  
Minor firmware version.

`uint8_t unused[2]`

`struct CeptonSensorInformation::[anonymous] formal_firmware_version`  
Firmware version struct.

`float last_reported_temperature`  
[celsius].

`float last_reported_humidity`  
[%].

`float last_reported_age`  
[hours].

`float measurement_period`  
Time between measurements [seconds].

`int64_t ptp_ts`  
PTP time [microseconds].

`uint8_t gps_ts_year`  
(0-99) (e.g. 2017 -> 17)

`uint8_t gps_ts_month`  
(1-12)

`uint8_t gps_ts_day`  
(1-31)

`uint8_t gps_ts_hour`  
(0-23)

`uint8_t gps_ts_min`  
(0-59)

`uint8_t gps_ts_sec`  
(0-59)

`uint8_t return_count`  
Number of returns per measurement.

`uint8_t segment_count`  
Number of image segments.

```

uint32_t flags
    Bit flags.

uint32_t is_mocked
    Created by capture replay.

uint32_t is_pps_connected
    GPS PPS is available.

uint32_t is_nmea_connected
    GPS NMEA is available.

uint32_t is_ptp_connected
    PTP is available.

uint32_t is_calibrated
    Calibration loaded.

uint32_t is_over_heated
    Hit temperature limit.

uint32_t is_sync_firing_enabled
    Sync fire enabled (disabled by default).

union CeptonSensorInformation::[anonymous] [anonymous]

```

### 5.3.2 Methods

```

int cepton_is_vista (CeptonSensorModel model)
    Returns whether sensor model is of the form VISTA_*.

int cepton_is_hr80 (CeptonSensorModel model)
    Returns whether sensor model is of the form HR80_*.

int cepton_is_vista (CeptonSensorModel model)
    Returns whether sensor model is of the form VISTA_*.

size_t cepton_sdk_get_n_sensors ()
    Get number of sensors attached. Use to check for new sensors. Sensors are not deleted until deinitialization.

CeptonSensorErrorCode cepton_sdk_get_sensor_handle_by_serial_number (uint64_t serial_number,
                                                                    CeptonSensorHandle
                                                                    *const handle)

    Looks up sensor handle by serial number.
    Returns error if sensor not found.

```

#### Parameters

- `serial_number`: Sensor serial number.
- `handle`: Sensor handle.

```

CeptonSensorErrorCode cepton_sdk_get_sensor_information_by_index (size_t idx, struct
                                                                    CeptonSensorIn-
                                                                    formation *const
                                                                    info)

    Returns sensor information by sensor index.

```

Useful for getting information for all sensors. Valid indices are in range [0, `cepton_sdk_get_n_sensors()`).

Returns error if index invalid.

#### Parameters

- `idx`: Sensor index. Returns error if invalid.
- `info`: Sensor information.

*CeptonSensorErrorCode* **cepton\_sdk\_get\_sensor\_information** (*CeptonSensorHandle* `handle`,  
**struct** *CeptonSensorInformation* \***const** `info`)

Returns sensor information by sensor handle.

#### Parameters

- `handle`: Sensor handle. Returns error if invalid.
- `info`: Sensor information.

## 5.4 Points

### 5.4.1 Types

**struct CeptonSensorImagePoint**

Point in pinhole image coordinates (focal length = 1).

To convert to 3d point, use `cepton_sdk::util::convert_sensor_image_point_to_point`.

#### Public Members

`int64_t` **timestamp**  
 Unix time [microseconds].

`float` **image\_x**  
 x image coordinate.

`float` **distance**  
 Distance [meters].

`float` **image\_z**  
 z image coordinate.

`float` **intensity**  
 Diffuse reflectance (normal: [0-1], retroreflective: >1).

*CeptonSensorReturnType* **return\_type**  
 Return type flags.

`uint8_t` **flags**  
 Bit flags.

`uint8_t` **valid**  
 If false, then distance and intensity are invalid.

```
uint8_t saturated
    If true, then intensity is invalid, and the distance is innacurate.
union CeptonSensorImagePoint::[anonymous] [anonymous]
uint8_t reserved[2]
```

## 5.4.2 Methods

```
typedef void (*FpCeptonSensorImageDataCallback)(CeptonSensorHandle handle, size_t
n_points, const struct CeptonSen-
sorImagePoint *c_points, void *user_data)
```

Callback for receiving image points.

Set the frame options to control the callback rate.

### Parameters

- handle: Sensor handle.
- n\_points: Points array size.
- c\_points: Points array. Owned by SDK.
- user\_data: User instance pointer.

```
CeptonSensorErrorCode cepton_sdk_listen_image_frames (FpCeptonSensorImageDataCallback
cb, void *const user_data)
```

Sets image frame callback.

Returns points at frequency specified by `cepton_sdk::FrameOptions::mode`. Each frame contains all possible points (use `cepton_sdk::SensorImagePoint::valid` to filter points). Points are ordered by measurement, segment, and return:

```
measurement_count = n_points / (segment_count * return_count)
idx = ((i_measurement) * segment_count + i_segment) * return_count + i_return
```

Returns error if callback already registered.

```
CeptonSensorErrorCode cepton_sdk_unlisten_image_frames ()
Clears image frame callback.
```

## 5.5 Capture Replay

PCAP capture file replay. Functions are not thread safe, and should only be called from the main thread.

```
int cepton_sdk_capture_replay_is_open ()
Returns whether capture replay is open.
```

```
CeptonSensorErrorCode cepton_sdk_capture_replay_open (const char *const path)
Opens capture replay.
```

Must be called before any other replay functions listed below.

### Parameters

- path: Path to PCAP capture file.

```
CeptonSensorErrorCode cepton_sdk_capture_replay_close ()
Closes capture replay.
```

**const char \*cepton\_sdk\_capture\_replay\_get\_filename ()**

Returns capture replay file name.

**int64\_t cepton\_sdk\_capture\_replay\_get\_start\_time ()**

Returns capture start Unix timestamp [microseconds].

**float cepton\_sdk\_capture\_replay\_get\_position ()**

Returns capture file position [seconds].

**float cepton\_sdk\_capture\_replay\_get\_length ()**

Returns capture file length [seconds].

**int cepton\_sdk\_capture\_replay\_is\_end ()**

Returns whether at end of capture file.

This is only relevant when using `resume_blocking` methods.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_seek (float position)**

Seek to capture file position [seconds].

### Parameters

- `position`: Seek position in range [0.0, capture length).

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_set\_enable\_loop (int enable\_loop)**

Sets capture replay looping.

If enabled, replay will automatically rewind at end.

**int cepton\_sdk\_capture\_replay\_get\_enable\_loop ()**

Returns whether capture replay looping is enabled.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_set\_speed (float speed)**

Sets speed multiplier for asynchronous replay.

**float cepton\_sdk\_capture\_replay\_get\_speed ()**

Returns capture replay speed.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_resume\_blocking\_once ()**

Replay next packet in current thread without sleeping.

Pauses replay thread if running.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_resume\_blocking (float duration)**

Replay multiple packets synchronously.

No sleep between packets. Pauses replay thread if running.

### Parameters

- `duration`: Duration to replay. Must be non-negative.

**int cepton\_sdk\_capture\_replay\_is\_running ()**

Returns true if replay thread is running.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_resume ()**

Resumes asynchronous replay thread.

Packets are replayed in realtime. Replay thread sleeps in between packets.

*CeptonSensorErrorCode* **cepton\_sdk\_capture\_replay\_pause ()**

Pauses asynchronous replay thread.



## 5.6 Networking

Network callback for debugging.

### 5.6.1 Types

```
typedef void (*FpCeptonNetworkReceiveCallback) (CeptonSensorHandle handle, int64_t
                                                timestamp, const uint8_t *buffer, size_t
                                                buffer_size, void *user_data)
```

Callback for receiving network packets.

Returns error if callback already set.

#### Parameters

- handle: Sensor handle.
- timestamp: Packet Unix timestamp [microseconds].
- buffer: Packet bytes.
- buffer\_size: Buffer size.
- user\_data: User instance pointer.

### 5.6.2 Methods

```
CeptonSensorErrorCode cepton_sdk_listen_network_packet (FpCeptonNetworkReceiveCallback
                                                         cb, void *const user_data)
```

Sets network packets callback.

For internal use.

Returns error if callback already registered.

#### Parameters

- cb: Callback.
- user\_data: User instance pointer.

```
CeptonSensorErrorCode cepton_sdk_unlisten_network_packet ()
```

Clears network packet callback.

## 5.7 Serial

Serial callback. Primarily used for receiving data from GPS/INS attached to sensor.

### 5.7.1 Types

```
typedef void (*FpCeptonSerialReceiveCallback) (CeptonSensorHandle handle, const char
                                                *str, void *user_data)
```

Callback for receiving serial data (e.g. NMEA).

#### Parameters

- `handle`: Sensor handle.
- `str`: Serial line string. Owned by SDK.
- `user_data`: User instance pointer.

## 5.7.2 Methods

*CeptonSensorErrorCode* **cepton\_sdk\_listen\_serial\_lines** (*FpCeptonSerialReceiveCallback* `cb`,  
void \***const** `user_data`)

Sets serial line callback.

Useful for listening to NMEA data from GPS attached to sensor. Each callback contains 1 line of serial data (including newline characters).

Returns error if callback already registered.

### Parameters

- `cb`: Callback.
- `user_data`: User instance pointer.

*CeptonSensorErrorCode* **cepton\_sdk\_unlisten\_serial\_lines** ()  
Clears serial line callback.

## 5.8 C++

High level C++ API for prototyping (`cepton_sdk.hpp`, `cepton_sdk_api.hpp`). Methods are agnostic to live/replay mode.

`bool cepton_sdk::api::is_live()`  
Returns whether capture replay is not open.

`bool cepton_sdk::api::is_end()`  
Returns whether capture replay is at the end and enable loop is false.

`int64_t cepton_sdk::api::get_time()`  
Returns live or capture replay time.

*SensorError* `cepton_sdk::api::wait` (float `t_length` = -1.0f)  
Sleeps or resumes capture replay for duration.  
If `t_length` < 0, then waits forever.

### 5.8.1 Errors

**CEPTON\_PROCESS\_ERROR** (code)  
Add context to error.

**CEPTON\_CHECK\_ERROR** (code)  
If error, raise.

**CEPTON\_LOG\_ERROR** (code)  
If error, print.

**CEPTON\_RETURN\_ERROR** (code)  
If error, return.

## 5.8.2 Setup

*SensorError* cepton\_sdk::api::**initialize** (Options *options* = create\_options(), **const** std::string &capture\_path = "", bool enable\_wait = false)

Initialize SDK and optionally starts capture replay.

If enable\_wait is true, waits a few seconds to initialize sensors.

*SensorError* cepton\_sdk::api::**open\_replay** (**const** std::string &capture\_path, bool enable\_wait = false)

Opens capture replay.

If enable\_wait is true, replays a few seconds to initialize sensors.

bool cepton\_sdk::api::**has\_control\_flags** (Control mask)

Returns whether indicated control flags are set.

*SensorError* cepton\_sdk::api::**enable\_control\_flags** (Control mask)

Enables indicated control flags.

*SensorError* cepton\_sdk::api::**disable\_control\_flags** (Control mask)

Disables indicated control flags.

**class** **SensorErrorCallback** : **public** cepton\_sdk::util::Callback<SensorHandle, **const** *SensorError*>

Callback for sensor errors.

### Public Static Functions

**static** void **global\_on\_callback** (SensorHandle handle, SensorErrorCode error\_code, **const** char \*error\_msg, **const** void \*const error\_data, size\_t error\_data\_size, void \*const instance)

**class** **SensorImageFrameCallback** : **public** cepton\_sdk::util::Callback<SensorHandle, std::size\_t, **const** SensorImagePoint>

Callback for image frames.

Must call initialize before use.

### Public Functions

**~SensorImageFrameCallback** ()

*SensorImageFrameCallback* class destructor.

*SensorError* **initialize** ()

Initializes *SensorImageFrameCallback* object.

*SensorError* **deinitialize** ()

Deinitializes *SensorImageFrameCallback* object.

bool **is\_initialized** () **const**

Returns true if *SensorImageFrameCallback* is initialized.

## 5.8.3 Sensors

bool cepton\_sdk::api::**has\_sensor\_by\_serial\_number** (uint64\_t serial\_number)

Returns whether SDK has sensor with serial number.

*SensorError* cepton\_sdk::api::get\_sensor\_information\_by\_serial\_number (uint64\_t *serial\_number*, SensorInformation &*info*)

Returns sensor information by serial number.

Returns error if sensor not found.

std::vector<uint64\_t> cepton\_sdk::api::get\_sensor\_serial\_numbers ()

Returns serial numbers for all sensors.

## 5.9 Utilities

Utility functions and classes for prototyping (*cepton\_sdk\_util.hpp*).

### 5.9.1 Common

int64\_t cepton\_sdk::util::get\_timestamp\_usec ()

Returns current unix timestamp [microseconds].

This is the timestamp format used by all sdk functions.

### 5.9.2 Points

**struct SensorPoint**

3d point class.

Can't subclass from SensorImagePoint, needs to be POD.

#### Public Members

int64\_t **timestamp**

Unix time [microseconds].

float **image\_x**

x image coordinate.

float **distance**

Distance [meters].

float **image\_z**

z image coordinate.

float **intensity**

Diffuse reflectance.

CeptonSensorReturnType **return\_type**

Strongest or farthest return.

uint8\_t **flags**

Bit flags.

uint8\_t **valid**

If false, then the distance and intensity are invalid.

uint8\_t **saturated**

If `true`, then the intensity is invalid. Also, the distance is valid, but inaccurate.

**union** cepton\_sdk::util::*SensorPoint*::[anonymous] [anonymous]

float **x**

x cartesian coordinate

float **y**

y cartesian coordinate

float **z**

z cartesian coordinate

```
void cepton_sdk::util::convert_sensor_image_point_to_point (const    SensorImage-
                                                             Point    &image_point,
                                                             SensorPoint &point)
```

Convenience method to convert cepton\_sdk::SensorImagePoint to cepton\_sdk::SensorPoint.

### 5.9.3 Callbacks

**template** <typename... *TArgs*>

**class** **Callback**

Expands SDK callback functionality.

Allows for multiple callbacks to be registered. Allows for registering lambdas and member functions. See `samples/basic.cpp`.

#### Public Functions

void **clear** ()

Clear all listeners.

*SensorError* **listen** (const std::function<void> *TArgs*...  
> &*func*, uint64\_t \*const *id* = nullptr) Register std::function.

#### Parameters

- *func*: *Callback* function.
- *id*: Identifier used for `unlisten`.

**template** <typename *TClass*>

*SensorError* **listen** (*TClass* \*const *instance*, MemberFunction<*TClass*, *TArgs*...> *func*, uint64\_t  
\*const *id* = nullptr)

Register instance member function.

#### Parameters

- *instance*: Parent class instance pointer.
- *func*: *Callback* function pointer.
- *id*: Identifier used for `unlisten`.

*SensorError* **unlisten** (uint64\_t *id*)

Unregister function.

### Parameters

- `id`: Identifier returned by `listen`.

void **operator()** (TArgs... *args*) **const**

Emit callback.

Calls all registered functions with *args*.

### Public Static Functions

**static** void **global\_on\_callback** (TArgs... *args*, void \***const** *instance*)

Used for registering as c callback.

## 5.9.4 Frames

**class** **FrameDetector** : **public** cepton\_sdk::util::FrameDetectorBase

Detects frames in streaming sensor data.

### Public Functions

**FrameDetector** (**const** SensorInformation &*sensor\_info*)

*FrameDetector* class constructor passing in SensorInformation.

**const** FrameOptions &**get\_options** () **const**

Returns frame options for *FrameDetector*.

*SensorError* **set\_options** (**const** FrameOptions &*options*)

Set frame options.

void **reset** ()

Completely resets detector.

Only use if also clearing points accumulator.

bool **update** (int64\_t *idx*, **const** SensorImagePoint &*point*)

Returns true if frame found.

Automatically resets after frame is found.

**class** **FrameAccumulator**

Accumulates image points, and emits frames to callback.

See `samples/frame.cpp`.

### Public Functions

**FrameAccumulator** (**const** SensorInformation &*sensor\_info*)

*FrameAccumulator* class constructor passing in SensorInformation.

FrameOptions **get\_options** () **const**

Return frame options for *FrameAccumulator*.

*SensorError* **set\_options** (**const** FrameOptions &*options*)

Set options for *FrameAccumulator*.

void **clear** ()

void **add\_points** (std::size\_t *n\_points*, **const** SensorImagePoint \***const** *image\_points*)

## Public Members

*Callback*<std::size\_t, **const** SensorImagePoint \*> **callback**

Frames callback.

## 5.9.5 Organizer

**struct OrganizedCloud**

The *OrganizedCloud* struct An organized version of the cepton point cloud.

## Public Functions

int **getIndex** (int *row*, int *col*, int *n\_return*)

getIndex Returns the index of the point corresponding to the inputted row, col and return number.

### Return

### Parameters

- *row*: Row index
- *col*: Col index
- *n\_return*: Return index

## Public Members

int64\_t **timestamp\_start**

timestamp\_start The time of the oldest point in the cloud

int64\_t **timestamp\_end**

timestamp\_end The time of the newest point in the cloud

int **height**

height Height of the cloud. Represents how many rows there are in the cloud

int **width**

width Width of the cloud. Represents how many columns there are in the cloud

int **n\_returns**

n\_returns Number of return represented by the cloud.

std::vector<*CellInfo*> **info\_cells**

info\_cells Vector of cell info which provide information about the matching points

std::vector<*CeptonSensorImagePoint*> **points**

points Vector of organized points. Stored in Return, Row, Col order. So to get a point at row 10, col 15, return 1 would be points[(row \* width

- col) n\_returns + return

**struct CellInfo**

The *CellInfo* struct.

**Public Members**

bool **occupied\_cell** = false

occupied\_cell Is the cell at this index occupied with a point. If false can't assume this represents free space.

int **original\_index** = -1

original\_index Index of the point that was used to generate the organized point. Can be used to match back with original data if required. Should only be use if occupied\_cell is true.

**class Organizer**

The *Organizer* class Performs organization on cepton unorganized points. Creates an angular grid, places each point within that grid and outputs a point for each location in the grid in a row/col format. Thread safe. Defaults to a 0.4deg spaced grid.

**Public Types****enum OrganizerMode**

Values:

**RECENT**

Output the most recent point from the frame that fell within the grid

**CENTER**

Output the center of the grid. Uses median point distance.

**Public Functions**

**Organizer** (cepton\_sdk::SensorInformation *sensor\_info*)

*Organizer*.

**Parameters**

- *sensor\_info*: Sensor info for organizer. Used to set min/max angles

void **organize\_points** (const int *num\_points\_in*, const int *n\_returns*, const *CeptonSensorImagePoint* \*const *unorganized\_points*, cepton\_sdk::util::OrganizedCloud &*organized\_points*)

*organize\_points*

**Parameters**

- *num\_points\_in*: Number of unorganized points
- *n\_returns*: Number of returns
- *unorganized\_points*: Unorganized points to proces
- *organized\_points*: Points in organized form

void **mode** (*OrganizerMode* *mode*)

*mode*



**Parameters**

- **mode**: Change the mode of the organizer. [RECENT] Points are the most recent which fill within the grid. [CENTER] Points outputted are at the center of the grid. More even spacing but less accurate.

void **binSize** (float *bin\_size*)

binSize Change the bin size of the organizer

**Parameters**

- **bin\_size**: The horizontal and vertical bin size to set. In radians

void **settings** (*OrganizerSettings* *organizer\_settings*)

settings

**Parameters**

- **organizer\_settings**: Change organizer settings

*Organizer::OrganizerSettings* **settings** ()

settings

**Return** The settings the organizer is using

**struct OrganizerSettings**

**Public Members**

float **horizontal\_range\_radians** = to\_radians(70.f)

float **vertical\_range\_radians** = to\_radians(30.f)

float **horizontal\_bin\_size\_radians** = to\_radians(0.4f)

float **vertical\_bin\_size\_radians** = to\_radians(0.4f)

*OrganizerMode* **mode** = *OrganizerMode::RECENT*



## Symbols

`_CeptonSDKControl` (C++ type), 28  
`_CeptonSDKFrameMode` (C++ type), 28  
`_CeptonSensorErrorCode` (C++ type), 26  
`_CeptonSensorModel` (C++ type), 31

## C

`CEPTON_CHECK_ERROR` (C macro), 38  
`CEPTON_ERROR_ALREADY_INITIALIZED` (C++ enumerator), 26  
`CEPTON_ERROR_COMMUNICATION` (C++ enumerator), 26  
`CEPTON_ERROR_CORRUPT_FILE` (C++ enumerator), 26  
`CEPTON_ERROR_EOF` (C++ enumerator), 27  
`CEPTON_ERROR_FILE_IO` (C++ enumerator), 26  
`CEPTON_ERROR_GENERIC` (C++ enumerator), 26  
`CEPTON_ERROR_INVALID_ARGUMENTS` (C++ enumerator), 26  
`CEPTON_ERROR_INVALID_FILE_TYPE` (C++ enumerator), 26  
`CEPTON_ERROR_NOT_INITIALIZED` (C++ enumerator), 26  
`CEPTON_ERROR_NOT_OPEN` (C++ enumerator), 27  
`CEPTON_ERROR_OUT_OF_MEMORY` (C++ enumerator), 26  
`CEPTON_ERROR_SDK_VERSION_MISMATCH` (C++ enumerator), 26  
`CEPTON_ERROR_SENSOR_NOT_FOUND` (C++ enumerator), 26  
`CEPTON_ERROR_TOO_MANY_CALLBACKS` (C++ enumerator), 26  
`CEPTON_FAULT_ABNORMAL_FOV` (C++ enumerator), 27  
`CEPTON_FAULT_ABNORMAL_FRAME_RATE` (C++ enumerator), 27  
`CEPTON_FAULT_DETECTOR_MALFUNCTION` (C++ enumerator), 27  
`CEPTON_FAULT_EXTREME_ACCELERATION` (C++ enumerator), 27  
`CEPTON_FAULT_EXTREME_HUMIDITY` (C++ enumerator), 27

`CEPTON_FAULT_EXTREME_TEMPERATURE` (C++ enumerator), 27  
`CEPTON_FAULT_INTERNAL` (C++ enumerator), 27  
`CEPTON_FAULT_LASER_MALFUNCTION` (C++ enumerator), 27  
`CEPTON_FAULT_MOTOR_MALFUNCTION` (C++ enumerator), 27  
`cepton_get_error_code_name` (C++ function), 27  
`cepton_is_error_code` (C++ function), 27  
`cepton_is_fault_code` (C++ function), 27  
`cepton_is_hr80` (C++ function), 33  
`cepton_is_vista` (C++ function), 33  
`CEPTON_LOG_ERROR` (C macro), 38  
`CEPTON_PROCESS_ERROR` (C macro), 38  
`CEPTON_RETURN_ERROR` (C macro), 38  
`cepton_sdk::api::disable_control_flags` (C++ function), 39  
`cepton_sdk::api::enable_control_flags` (C++ function), 39  
`cepton_sdk::api::get_sensor_information_by_serial_number` (C++ function), 39  
`cepton_sdk::api::get_sensor_serial_numbers` (C++ function), 40  
`cepton_sdk::api::get_time` (C++ function), 38  
`cepton_sdk::api::has_control_flags` (C++ function), 39  
`cepton_sdk::api::has_sensor_by_serial_number` (C++ function), 39  
`cepton_sdk::api::initialize` (C++ function), 39  
`cepton_sdk::api::is_end` (C++ function), 38  
`cepton_sdk::api::is_live` (C++ function), 38  
`cepton_sdk::api::open_replay` (C++ function), 39  
`cepton_sdk::api::SensorErrorCallback` (C++ class), 39  
`cepton_sdk::api::SensorErrorCallback::global_on_callback` (C++ function), 39  
`cepton_sdk::api::SensorImageFrameCallback` (C++ class), 39  
`cepton_sdk::api::SensorImageFrameCallback::~SensorImageFrameCallback`

(C++ function), 39

cepton\_sdk::api::SensorImageFrameCallback::deinit (C++ member), 43

(C++ function), 39

cepton\_sdk::api::SensorImageFrameCallback::init (C++ function), 43

(C++ function), 39

cepton\_sdk::api::SensorImageFrameCallback::is\_initialized (C++ function), 42

(C++ function), 39

cepton\_sdk::api::wait (C++ function), 38

cepton\_sdk::FpSensorErrorCallback (C++ type), 29

cepton\_sdk::SensorError (C++ class), 25

cepton\_sdk::SensorError::~~SensorError (C++ function), 25

cepton\_sdk::SensorError::code (C++ function), 25

cepton\_sdk::SensorError::ignore (C++ function), 25

cepton\_sdk::SensorError::is\_error (C++ function), 26

cepton\_sdk::SensorError::is\_fault (C++ function), 26

cepton\_sdk::SensorError::msg (C++ function), 25

cepton\_sdk::SensorError::name (C++ function), 26

cepton\_sdk::SensorError::operator bool (C++ function), 26

cepton\_sdk::SensorError::operator SensorErrorCode (C++ function), 26

cepton\_sdk::SensorError::operator= (C++ function), 25

cepton\_sdk::SensorError::SensorError (C++ function), 25

cepton\_sdk::SensorError::used (C++ function), 25

cepton\_sdk::SensorError::what (C++ function), 25

cepton\_sdk::util::Callback (C++ class), 41

cepton\_sdk::util::Callback::clear (C++ function), 41

cepton\_sdk::util::Callback::global\_on\_callback (C++ function), 42

cepton\_sdk::util::Callback::listen (C++ function), 41

cepton\_sdk::util::Callback::operator () (C++ function), 42

cepton\_sdk::util::Callback::unlisten (C++ function), 41

cepton\_sdk::util::convert\_sensor\_image\_point\_to\_host\_point (C++ function), 41

cepton\_sdk::util::FrameAccumulator (C++ class), 42

cepton\_sdk::util::FrameAccumulator::add\_points (C++ enumerator), 44

(C++ function), 43

cepton\_sdk::util::FrameAccumulator::callback (C++ member), 43

cepton\_sdk::util::FrameAccumulator::clear (C++ function), 42

cepton\_sdk::util::FrameAccumulator::FrameAccumulator (C++ function), 42

cepton\_sdk::util::FrameAccumulator::get\_options (C++ function), 42

cepton\_sdk::util::FrameAccumulator::set\_options (C++ function), 42

cepton\_sdk::util::FrameDetector (C++ class), 42

cepton\_sdk::util::FrameDetector::FrameDetector (C++ function), 42

cepton\_sdk::util::FrameDetector::get\_options (C++ function), 42

cepton\_sdk::util::FrameDetector::reset (C++ function), 42

cepton\_sdk::util::FrameDetector::set\_options (C++ function), 42

cepton\_sdk::util::FrameDetector::update (C++ function), 42

cepton\_sdk::util::get\_timestamp\_usec (C++ function), 40

cepton\_sdk::util::OrganizedCloud (C++ class), 43

cepton\_sdk::util::OrganizedCloud::CellInfo (C++ class), 43

cepton\_sdk::util::OrganizedCloud::CellInfo::occupied (C++ member), 44

cepton\_sdk::util::OrganizedCloud::CellInfo::original (C++ member), 44

cepton\_sdk::util::OrganizedCloud::getIndex (C++ function), 43

cepton\_sdk::util::OrganizedCloud::height (C++ member), 43

cepton\_sdk::util::OrganizedCloud::info\_cells (C++ member), 43

cepton\_sdk::util::OrganizedCloud::n\_returns (C++ member), 43

cepton\_sdk::util::OrganizedCloud::points (C++ member), 43

cepton\_sdk::util::OrganizedCloud::timestamp\_end (C++ member), 43

cepton\_sdk::util::OrganizedCloud::timestamp\_start (C++ member), 43

cepton\_sdk::util::OrganizedCloud::width (C++ member), 43

cepton\_sdk::util::Organizer (C++ class), 44

cepton\_sdk::util::Organizer::binSize (C++ function), 45

cepton\_sdk::util::Organizer::CENTER (C++ member), 45

cepton\_sdk::util::Organizer::mode (C++ member), 45

*function*), 44  
 cepton\_sdk::util::Organizer::organize\_points (C++ function), 44  
 cepton\_sdk::util::Organizer::Organizer (C++ function), 44  
 cepton\_sdk::util::Organizer::OrganizerMode (C++ type), 44  
 cepton\_sdk::util::Organizer::OrganizerSettings (C++ class), 45  
 cepton\_sdk::util::Organizer::OrganizerSettings::display\_get\_radians\_time (C++ member), 45  
 cepton\_sdk::util::Organizer::OrganizerSettings::display\_get\_radians (C++ member), 45  
 cepton\_sdk::util::Organizer::OrganizerSettings::display\_is\_open (C++ member), 45  
 cepton\_sdk::util::Organizer::OrganizerSettings::display\_is\_radians (C++ member), 45  
 cepton\_sdk::util::Organizer::OrganizerSettings::display\_is\_radians (C++ member), 45  
 cepton\_sdk::util::Organizer::RECENT (C++ enumerator), 44  
 cepton\_sdk::util::Organizer::settings (C++ function), 45  
 cepton\_sdk::util::SensorPoint (C++ class), 40  
 cepton\_sdk::util::SensorPoint::distance (C++ member), 40  
 cepton\_sdk::util::SensorPoint::flags (C++ member), 40  
 cepton\_sdk::util::SensorPoint::image\_x (C++ member), 40  
 cepton\_sdk::util::SensorPoint::image\_z (C++ member), 40  
 cepton\_sdk::util::SensorPoint::intensity (C++ member), 40  
 cepton\_sdk::util::SensorPoint::return\_type (C++ member), 40  
 cepton\_sdk::util::SensorPoint::saturated (C++ member), 40  
 cepton\_sdk::util::SensorPoint::timestamp (C++ member), 40  
 cepton\_sdk::util::SensorPoint::valid (C++ member), 40  
 cepton\_sdk::util::SensorPoint::x (C++ member), 41  
 cepton\_sdk::util::SensorPoint::y (C++ member), 41  
 cepton\_sdk::util::SensorPoint::z (C++ member), 41  
 cepton\_sdk::util::SensorPoint::[anonymous] (C++ member), 41  
 cepton\_sdk\_capture\_replay\_close (C++ function), 35  
 cepton\_sdk\_capture\_replay\_get\_enable\_loop (C++ function), 36  
 cepton\_sdk\_capture\_replay\_get\_filename (C++ function), 36  
 cepton\_sdk\_capture\_replay\_get\_length (C++ function), 36  
 cepton\_sdk\_capture\_replay\_get\_position (C++ function), 36  
 cepton\_sdk\_capture\_replay\_get\_speed (C++ function), 36  
 cepton\_sdk\_capture\_replay\_get\_radians\_time (C++ function), 36  
 cepton\_sdk\_capture\_replay\_get\_radians (C++ function), 36  
 cepton\_sdk\_capture\_replay\_is\_open (C++ function), 35  
 cepton\_sdk\_capture\_replay\_is\_radians (C++ function), 36  
 cepton\_sdk\_capture\_replay\_is\_radians (C++ function), 35  
 cepton\_sdk\_capture\_replay\_pause (C++ function), 36  
 cepton\_sdk\_capture\_replay\_resume (C++ function), 36  
 cepton\_sdk\_capture\_replay\_resume\_blocking (C++ function), 36  
 cepton\_sdk\_capture\_replay\_resume\_blocking\_once (C++ function), 36  
 cepton\_sdk\_capture\_replay\_seek (C++ function), 36  
 cepton\_sdk\_capture\_replay\_set\_enable\_loop (C++ function), 36  
 cepton\_sdk\_capture\_replay\_set\_speed (C++ function), 36  
 cepton\_sdk\_clear (C++ function), 30  
 CEPTON\_SDK\_CONTROL\_DISABLE\_DISTANCE\_CLIP (C++ enumerator), 28  
 CEPTON\_SDK\_CONTROL\_DISABLE\_IMAGE\_CLIP (C++ enumerator), 28  
 CEPTON\_SDK\_CONTROL\_DISABLE\_NETWORK (C++ enumerator), 28  
 CEPTON\_SDK\_CONTROL\_ENABLE\_CROSSTALK\_FILTER (C++ enumerator), 28  
 CEPTON\_SDK\_CONTROL\_ENABLE\_MULTIPLE\_RETURNS (C++ enumerator), 28  
 CEPTON\_SDK\_CONTROL\_ENABLE\_STRAY\_FILTER (C++ enumerator), 28  
 CEPTON\_SDK\_CONTROL\_HOST\_TIMESTAMPS (C++ enumerator), 28  
 cepton\_sdk\_create\_frame\_options (C++ function), 29  
 cepton\_sdk\_create\_options (C++ function), 29  
 cepton\_sdk\_deinitialize (C++ function), 30  
 CEPTON\_SDK\_FRAME\_COVER (C++ enumerator), 28  
 CEPTON\_SDK\_FRAME\_CYCLE (C++ enumerator), 29

CEPTON\_SDK\_FRAME\_MODE\_MAX (C++ *enumerator*),  
     29  
 CEPTON\_SDK\_FRAME\_STREAMING (C++ *enumera-  
 tor*), 28  
 CEPTON\_SDK\_FRAME\_TIMED (C++ *enumerator*), 28  
 cepton\_sdk\_get\_control\_flags (C++ *func-  
 tion*), 30  
 cepton\_sdk\_get\_error (C++ *function*), 27  
 cepton\_sdk\_get\_frame\_length (C++ *function*),  
     31  
 cepton\_sdk\_get\_frame\_mode (C++ *function*), 31  
 cepton\_sdk\_get\_n\_sensors (C++ *function*), 33  
 cepton\_sdk\_get\_port (C++ *function*), 30  
 cepton\_sdk\_get\_sensor\_handle\_by\_serial\_number  
     (C++ *function*), 33  
 cepton\_sdk\_get\_sensor\_information (C++  
     *function*), 34  
 cepton\_sdk\_get\_sensor\_information\_by\_index  
     (C++ *function*), 33  
 cepton\_sdk\_get\_version\_major (C++ *func-  
 tion*), 30  
 cepton\_sdk\_get\_version\_minor (C++ *func-  
 tion*), 30  
 cepton\_sdk\_get\_version\_string (C++ *func-  
 tion*), 30  
 cepton\_sdk\_has\_control\_flag (C++ *function*),  
     30  
 cepton\_sdk\_initialize (C++ *function*), 30  
 cepton\_sdk\_listen\_image\_frames (C++ *func-  
 tion*), 35  
 cepton\_sdk\_listen\_network\_packet (C++  
     *function*), 37  
 cepton\_sdk\_listen\_serial\_lines (C++ *func-  
 tion*), 38  
 cepton\_sdk\_set\_control\_flags (C++ *func-  
 tion*), 30  
 cepton\_sdk\_set\_frame\_options (C++ *func-  
 tion*), 30  
 cepton\_sdk\_set\_port (C++ *function*), 30  
 cepton\_sdk\_unlisten\_image\_frames (C++  
     *function*), 35  
 cepton\_sdk\_unlisten\_network\_packet (C++  
     *function*), 37  
 cepton\_sdk\_unlisten\_serial\_lines (C++  
     *function*), 38  
 CEPTON\_SENSOR\_MODEL\_MAX (C++ *enumerator*),  
     31  
 CEPTON\_SUCCESS (C++ *enumerator*), 26  
 CeptonSDKControl (C++ *type*), 28  
 CeptonSDKFrameMode (C++ *type*), 28  
 CeptonSDKFrameOptions (C++ *class*), 29  
 CeptonSDKFrameOptions::length (C++ *mem-  
 ber*), 29  
 CeptonSDKFrameOptions::mode (C++ *member*),  
     29  
 CeptonSDKFrameOptions::signature (C++  
     *member*), 29  
 CeptonSDKOptions (C++ *class*), 29  
 CeptonSDKOptions::control\_flags (C++  
     *member*), 29  
 CeptonSDKOptions::frame (C++ *member*), 29  
 CeptonSDKOptions::port (C++ *member*), 29  
 CeptonSDKOptions::signature (C++ *member*),  
     29  
 CeptonSensorErrorCode (C++ *type*), 26  
 CeptonSensorHandle (C++ *type*), 31  
 CeptonSensorImagePoint (C++ *class*), 34  
 CeptonSensorImagePoint::distance (C++  
     *member*), 34  
 CeptonSensorImagePoint::flags (C++ *mem-  
 ber*), 34  
 CeptonSensorImagePoint::image\_x (C++  
     *member*), 34  
 CeptonSensorImagePoint::image\_z (C++  
     *member*), 34  
 CeptonSensorImagePoint::intensity (C++  
     *member*), 34  
 CeptonSensorImagePoint::reserved (C++  
     *member*), 35  
 CeptonSensorImagePoint::return\_type  
     (C++ *member*), 34  
 CeptonSensorImagePoint::saturated (C++  
     *member*), 34  
 CeptonSensorImagePoint::timestamp (C++  
     *member*), 34  
 CeptonSensorImagePoint::valid (C++ *mem-  
 ber*), 34  
 CeptonSensorImagePoint::[anonymous]  
     (C++ *member*), 35  
 CeptonSensorInformation (C++ *class*), 31  
 CeptonSensorInformation::firmware\_version  
     (C++ *member*), 32  
 CeptonSensorInformation::flags (C++ *mem-  
 ber*), 32  
 CeptonSensorInformation::formal\_firmware\_version  
     (C++ *member*), 32  
 CeptonSensorInformation::gps\_ts\_day  
     (C++ *member*), 32  
 CeptonSensorInformation::gps\_ts\_hour  
     (C++ *member*), 32  
 CeptonSensorInformation::gps\_ts\_min  
     (C++ *member*), 32  
 CeptonSensorInformation::gps\_ts\_month  
     (C++ *member*), 32  
 CeptonSensorInformation::gps\_ts\_sec  
     (C++ *member*), 32  
 CeptonSensorInformation::gps\_ts\_year  
     (C++ *member*), 32

CeptonSensorInformation::handle (C++ FpCeptonSerialReceiveCallback (C++ type),  
     *member*), 31 37  
 CeptonSensorInformation::is\_calibrated (C++ *member*), 33 **H**  
 CeptonSensorInformation::is\_mocked (C++ HR80T\_R2 (C++ *enumerator*), 31  
     *member*), 33 HR80W (C++ *enumerator*), 31  
 CeptonSensorInformation::is\_nmea\_connected (C++ *member*), 33 **S**  
 CeptonSensorInformation::is\_over\_heated SORA\_P60 (C++ *enumerator*), 31  
     (C++ *member*), 33 SORA\_P61 (C++ *enumerator*), 31  
 CeptonSensorInformation::is\_pps\_connected SORA\_P90 (C++ *enumerator*), 31  
     (C++ *member*), 33 **V**  
 CeptonSensorInformation::is\_ptp\_connected VISTA\_860\_GEN2 (C++ *enumerator*), 31  
     (C++ *member*), 33 VISTA\_H120 (C++ *enumerator*), 31  
 CeptonSensorInformation::last\_reported\_age VISTA\_P60 (C++ *enumerator*), 31  
     (C++ *member*), 32 VISTA\_P61 (C++ *enumerator*), 31  
 CeptonSensorInformation::last\_reported\_humidity VISTA\_P90 (C++ *enumerator*), 31  
     (C++ *member*), 32 VISTA\_X120 (C++ *enumerator*), 31  
 CeptonSensorInformation::last\_reported\_temperature VISTA\_X15 (C++ *enumerator*), 31  
     (C++ *member*), 32  
 CeptonSensorInformation::major (C++ *member*), 32  
 CeptonSensorInformation::measurement\_period (C++ *member*), 32  
 CeptonSensorInformation::minor (C++ *member*), 32  
 CeptonSensorInformation::model (C++ *member*), 32  
 CeptonSensorInformation::model\_name (C++ *member*), 32  
 CeptonSensorInformation::ptp\_ts (C++ *member*), 32  
 CeptonSensorInformation::reserved (C++ *member*), 32  
 CeptonSensorInformation::return\_count (C++ *member*), 32  
 CeptonSensorInformation::segment\_count (C++ *member*), 32  
 CeptonSensorInformation::serial\_number (C++ *member*), 31  
 CeptonSensorInformation::unused (C++ *member*), 32  
 CeptonSensorInformation::[anonymous] (C++ *member*), 33  
 CeptonSensorModel (C++ *type*), 31

**F**

FpCeptonNetworkReceiveCallback (C++ *type*),  
     37  
 FpCeptonSensorImageDataCallback (C++  
     *type*), 35